Notas sobre satisfiabilidad Ver 2.0

Juan Manuel Dato Ruiz

jumadaru@gmail.com

A día de hoy he tenido la oportunidad de enfrentar el programa que hice en Python 3.0 para ejecutar un algoritmo que conocía hace muchos años. Poco a poco he estado depurando mis conocimientos y he pretendido estar divulgando exclusivamente lo consolidado.

Debido a que confeccioné antes el algoritmo que el programa que lo ponía al límite, empezaron a aparecer errores y otros imprevistos de concepto que me obligan a tener que hacer según qué modificaciones, aunque parece que los pilares siguen firmes.

1. Supongamos que disponemos de dos fórmulas F(X) y G(Y) donde X e Y son tuplas de literales. Diremos que para cualquier X F(X)=Falso y que existe un Y donde G(Y)=True. Por tanto la fórmula H(X,Y) = F(X) and G(Y) = Falso. Esto nos lleva a que con el algoritmo presentado, tras evaluar G y sus literales, el algoritmo parará dando la impresión de que existe una solución (ya que la resolución del algoritmo no depende de Y). Esto indica que para determinar la satisfiabilidad se debe:
    a. Lanzar un supuesto en cada una de las cláusulas.
    b. Intentar terminar de encontrar un resultado completo.

    Es por ello que no podemos contentarnos con ver cómo nos obliga el algoritmo a lanzar otro supuesto, porque el que tengamos que volver a lanzar el algoritmo no implica que haya solución, pues es posible que la solución que debamos recorrer precise ser al menos planteada antes de poder rechazarla. Como este proceso no anida supuestos podemos mantener la polinimialidad.

2. El programa hecho en Python tenía por objeto demostrar que existía un código asociable al algoritmo, no obstante no descarto la posibilidad de implementarlo de una manera bien estructurada orientado a objetos; con el fin de crear las abstracciones oportunas y así mejorar la legibilidad y su continuidad.
3. Actualmente en 2.0 he encontrado bugs, a la hora de enfrentarlo a fórmulas de relevancia. Concretamente a la hora de tratar la siguiente posición activa; se eligió una política a la hora de implementar el algoritmo muy oscura y, para mayor gravedad, le quise incluir un mecanismo para calcular el número de casos. Desde mi punto de vista fracasaron ambos intentos, lo mejor será clarificar esa parte del código.
4. También encontré un error tipográfico y otro de mal uso de la captura de los casos. El error tipográfico se subsanará fácilmente, pero la captura de casos voy a desecharlo para la siguiente versión. Ni esta estructura es la más adecuada ni parece que esté bien enfocado, lo que haré será hacer que se acceda a una solución en principio diferente mediante una función de dispersión.

5. El programa hecho en Python es muy conservador. Concretamente, cada vez que tiene que estudiar la estructura suele clonarla, con su consecuente aumento del factor de costes temporales. Además, la estructura gasta una posición para albergar sólo cuatro posibles valores; bien podría gastar una posición para gastar 4^9 valores y así a las tablas se accedería más rápidamente, aprovechando la potencia de la máquina. Es por ello que el algoritmo en sí no está pensado para funcionar muy rápidamente, todo lo más, está bien condicionado para ser acelerado. Pero de todas formas hace falta algunos compromisos adicionales.
6. No descarto la posibilidad de crear un documento FAQ para cualquiera de mis aportaciones. Sin embargo para ello antes quiero que se use mi e-mail jumadaru@gmail.com, por coherencia. Al fin y al cabo mis reflexiones no tienen por qué ser compartidas, y cualquier aspecto del código es susceptible de poder ser explicado y defendido.
7. Huelga mencionar mi agradecimiento por tomar en consideración estas aportaciones que, como cualquier otra, debe ser objeto de crítica y mejora.